



VORTEX++ Class Reference Manual

May 30, 2017

Trifox Inc.

www.trifox.com



Trademarks

TRIMapp, TRImpl, TRIMqmr, TRIMreport, TRIMtools, GENESISsql, DesignVision, DVapp, DVreport, VORTEX, VORTEXcli, VORTEXc, VORTEXcobol, VORTEXperl, VORTEXjdbc, VORTEX++, VORTEXJava Edition, LIST Manager, VORTEXodbc, VORTEXnet, VORTEXclient/server, VORTEXaccelerator, VORTEXreplicator are all trademarks of Trifox, Inc.

All other brand and product names are trademarks or registered trademarks of their respective owners.

Copyright

The information contained in this document is subject to change without notice and does not represent a commitment by Trifox Inc. The software described in this document is furnished under a license agreement and may be used or copied only in accordance with the terms of the agreement. No part of this manual or software may be reproduced or transmitted in any form or by any means, electronic or mechanical (including photocopying and recording), or transferred to information storage and retrieval systems without the written permission of Trifox Inc.

Copyright © Trifox Inc. 1986-2017

All rights reserved.

Printed in the U.S.A.



Contents

Revisions vii

1 Overview

Overview 7

2 dbChannel

class dbChannel 8

Data Members 8

Constructors 8

Destructor 8

Exceptions 9

Member Functions 10

cancel 10

command 10

commit/rollback 11

sql 12

proc 12

execute 12

fetch 13

close 13

nextObject 13

skipObject 14

getBlob 14

putBlob 14

getRowsAffected 15

getCode/getMessage 15

set/getIn/OutDateMask 15

setUserErrorMap 15

userError 15

stmtCacheOn/Off 16

throwException 16

connect 16

release 16

3 dbChar/dbBinary

class dbChar:public dbObject 17

Data Members 17

Constructors 17

Destructor 17

Member Functions 17

4 dbDate

class dbDate : public dbObject 18

Data Members 18

Constructors 18

Destructor 18

- Exceptions 18
 - Member Functions 18
 - timestamp/today 18
 - toChar 18
- 5 dbDescriptor**
 - class dbDescriptor 19
 - Data Members 19
 - Constructors 19
 - Destructor 19
 - Member Functions 19
 - isLargeObject 19
- 6 dbException**
 - class dbException:public xmsg 20
 - Catching Messages 20
 - Data Members 20
 - Constructors 21
 - Member Functions 21
 - getType/getCode 21
 - getChannel 21
- 7 dbInt**
 - class dbInt:public dbObject 22
 - Data Members 22
 - Constructors 22
 - Destructor. 22
 - Member Functions 22
- 8 dbNumber**
 - class dbNumber:public dbObject 23
 - Data Members 23
 - Constructors 23
 - Destructor. 23
 - Member Functions 23
 - toInt 23
 - toInt8 24
 - toUInt 24
 - toUInt8 24
 - toFloat/toDouble 25
 - sqrt 25
- 9 dbObject**
 - class dbObject 26
 - Data Members 26
 - Constructors 26
 - Destructor. 26
 - Member Functions 26
 - getDatatype/getLength 26
 - getAddr 26
 - toChar 26
 - copyObject 27
- 10 dbStatement**
 - class dbStatement 28

Data Members	28
Constructors	28
Destructor	28
Exceptions	29
Member Functions	29
hasBlobs	29
execute	29
fetch	29
close	29
getBlob	29
putBlob	30
nextObject	30
skipObject	30
copyParam	30
setParam	30
setHostVar	31
numOutputCols	31
getColDesc	32
endOfScan	32
getChannel	32



Preface

Background

Trifox Inc. has been serving the relational database market since 1984 through consulting and the development of software products. In 1987, Trifox created SQL*QMX for Oracle. This easy-to-use, powerful querying and report writing tool, which is based on IBM's QMF, continues to be used at thousands of sites. In 1989, Trifox created TRIMtools, a family of application and reportwriting tools now known as DesignVision. DesignVision was developed in response to the OLTP requirements of several large application vendors.

Database Access

VORTEX is an integrated family of products that allows nearly any production application to access SQL data:

- On any or all of the major relational databases.
- Across networks.
- Across platforms.
- With a dramatic increase in the number of concurrent users.
- Without any additional hardware.

In a client/server or multi-tier configuration, VORTEX makes it possible for your SQL applications to access data on different platforms over one or more network configurations. Currently it supports only TCP/IP.

Inherent in this approach are services that allow production applications originally written for one relational database (such as Oracle) can access the same data on another database (such as Informix), even if it is spread across different databases.

VORTEX Precompilers for C and COBOL, as well as a variety of program interfaces, allow existing SQL programs to take full advantage of VORTEX services such as performance enhancement, transaction monitoring, and flat-file database access.

With VORTEXaccelerator in your configuration, you dramatically increase the number of concurrent users who can log on to a specific SQL production application. Your users experience faster performance and you won't have to change any programs or add any hardware.

Application and Report Development

DesignVision DVapp lets you design, generate, and maintain forms-based applications. You can easily port the pop-up windows, customizable menus and submenus, and

custom keyboard assignments, in fact the entire application, to Windows .NET, Unix, OpenVMS, or HTML5 with no extra effort.

The reportwriter, TRIMreport, lets you create simple reports quickly, or complex reports with absolute confidence in their power.

When you want to write stand-alone applications (including triggers) without a user interface, the TRIMpl 4GL language gives you the freedom you want. The procedural language has over 100 database-specific functions that help you write powerful applications in very little time.

Reaching Legacy Data

GENESISsql is a SQL processor that accesses low-level data sources such as ISAM, SDMS, ADABAS, RMS, and MicroFocus and makes the data accessible to VORTEX clients. You can add GENESIS data sources to a VORTEX system in a matter of days, simplifying what used to be an enormous task.

Conventions

Screen shots in this manual come from the Windows version of our software.

Trifox documentation uses the following conventions for communicating information:

Example	Describes
CHOOSE REPORT > [F3] >	Press [F3] on the CHOOSE REPORT menu and ...
Right-click	Clicking the right mouse button.
Left-click	Clicking the left mouse button.
<i>connect_string</i>	Replace italicized text with your own variable.
vtxnetd	Text in bold typewriter style represents strings that you type exactly as they appear in the manual.

Support

If you have a question about a TRIFOX product that is not answered in the documentation (paper or online), contact the Customer Support Services group at:

- support@trifox.com
- Trifox Customer Support Services
2959 Winchester Boulevard
Campbell, CA 95008
U.S.A.
- 408-796-1590

Revisions

August 1996

First release.



Chapter 1

Overview

Overview

VORTEX++ is a thin class library that provides a method of accessing the VORTEX API, which is a message-based interface. The interface is message-based, machine-independent, and supports three data types:

- CHAR/BINARY
- NUMBER
- DATETIME

Since the API is message based and machine-independent, it moves data very efficiently but isn't easy to use. VORTEX++ enhances usability by providing:

- A few simple classes that handle all database access.
- Convenient classes to handle the basic data types.
- Operator overloading.
- Conversion methods.

The main VORTEX++ classes are:

- *dbChannel* — Encapsulates a database connection. Each instance corresponds to an actual database connection.
- *dbStatement* — Encapsulates a SQL statement. Each instance corresponds to a logical VORTEX cursor.
- *dbDescriptor* — Describes an output column.
- *dbException* — The base class for all VORTEX++ exceptions.
- *dbObject* — The base class for all VORTEX++ data objects.



Chapter 2

dbChannel

class dbChannel

Encapsulates a database connection.

Data Members

```
int typeOfLastException;
```

`typeOfLastException` is initialized to zero (0) on instantiation and is never reset. Each time VORTEX throws an exception, its type is stored in this information variable. The possible values are:

1. `dbException`
2. `dbExceptError`
3. `dbExceptDB`
4. `dbExceptUser`
5. `dbExceptFatal`
6. `dbExceptVortex`
7. `dbExceptNet`

See `dbChannel.hxx` for more details.

Constructors

```
dbChannel ( const int maxLogicalCursors,  
            const int maxDbCursors,  
            const int maxColumns,  
            const int fetchBufferSize);
```

maxLogicalCursors Maximum number of logical cursors to be used. Set this to some high number, such as 512.

maxDbCursors Maximum number of actual database cursors to allocate. A good number is 1/8 of `maxLogicalCursors`.

maxColumns Maximum number of columns to be processed at any given time.

fetchBufferSize Size of the internal buffer used to pre-fetch data.

Destructor

A database release is automatically performed if a connection is active.

```
~dbChannel ();
```

Exceptions

The following exceptions can be thrown by dbChannel functions.

dbExceptionError

- Missing parameter, row: n, column: m
- Datatype mismatch, row: n, column: m
- Unknown BIND type n
- Invalid execute procedure state
- Data conversion failed
- Invalid method for procedure
- Invalid execute cursor state
- Invalid fetch cursor state
- Invalid SELECT statement
- Expected a dbBinary
- Expected a dbChar
- Expected a dbNumber
- Expected a dbDate
- Unexpected datatype n
- returnObject cannot be NULL

dbExceptDB

Any message that can be returned from the target DBMS

dbExceptUser

Any message that can be returned from the target DBMS and is remapped to a user defined error number

dbExceptFatal

No more logical cursors available

dbExceptVortex

See the Codes and Messages chapter of the VORTEX Installation and Usage Guide, VORTEX Error Messages section.

dbExceptNet

Any operating system network error message

Member Functions

These functions are listed in `dbChannel.hxx`.

cancel

Cancels any outstanding database request. Typically this function is called from a Ctrl-C handler when the user wants to interrupt a wild query.

```
void cancel();
```

command

Executes a database driver-specific command.

```
void command (dbStatement *theStatement,
              const int   commandCode,
              const char *commandParms);
```

theStatement NULL or, if a statement (cursor) command, a previously instantiated `dbStatement`. The parameter is ignored for commands that are session-oriented.

commandCode The command code.

commandParms A null-terminated string containing the command parameters. Multiple parameters must be blank-separated.

The following command are available: (see `dbChannel.hxx` for definitions.)

1. TDB_CMD_TIMEOUT
2. TDB_CMD_SINGLEPID
3. TDB_CMD_OOPT
4. TDB_CMD_USEDDB
5. TDB_CMD_LANGUAGE
6. TDB_CMD_RAW_DATETIME
7. TDB_CMD_VERSION
8. TDB_CMD_CHAR

9. TDB_CMD_LOCK
10. TDB_CMD_ONEBYTE
11. TDB_CMD_MULTISTMT
12. TDB_CMD_AUTOCOMMIT
13. TDB_CMD_DORMANT
14. TDB_CMD_GETLDA
15. TDB_CMD_RAISE
16. TDB_CMD_ERROR
17. TDB_CMD_TMP_RELEASE
18. TDB_CMD_WORM_AUTH
19. TDB_CMD_TIMEOUT_QUERY
20. TDB_CMD_TIMEOUT_FETCH
21. TDB_CMD_DYN_CURSOR
22. TDB_CMD_SYNTAX_LEVEL
23. TDB_CMD_RETURN_ROWID
24. TDB_CMD_NEW_BLOBS
25. TDB_CMD_RO_CURSOR
26. TDB_CMD_CURSOR_TYPE
27. TDB_CMD_TXN_ISOLEVEL
28. TDB_CMD_SCROLL
29. TDB_CMD_HOSTNAME
30. TDB_CMD_FQCOLNAME
31. TDB_CMD_FREE_LOCKS
32. TDB_CMD_NOUNICHAR
33. TDB_CMD_ZEROLEN_NULL
34. TDB_CMD_EXIO_RS
35. TDB_CMD_DRV_CONFIG

commit/rollback

Commits or rolls back a transaction and, optionally, starts a new read/write transaction.

```
void commit (const int startUpdateTrans);  
void rollback (const int startUpdateTrans);
```

startUpdateTrans If true then start a read/write transaction following the commit/rollback.

sql

Associates a SQL statement with a `dbStatement`. If the statement contains parameters in the Oracle format `:n`, then it's translated for the target database (example: `"?"` for DB2). No other translation is performed.

Use `dbStatement::copyParam` and `dbStatement::setParam` to populate the parameters.

```
void sql (dbStatement *theStatement,
         const char *sqlStatement);
void sql (dbStatement *theStatement,
         const char *sqlStatement,
         const int numParameters);
void sql (dbStatement *theStatement,
         const char *sqlStatement,
         const int numDimensions),
         const int numParameters)
```

theStatement	A previously instantiated <code>dbStatement</code> .
sqlStatement	A null-terminated string containing the SQL statement.
numParameters	Number of parameters in the SQL statement. If not specified, then the statement is assumed to have no parameters.
numDimensions	Number of dimensions of parameters in the SQL statement. If not specified and <code>numParameters</code> is specified, the value defaults to 1.

proc

Associates a stored procedure statement with a `dbStatement`. Note that not all databases support stored procedures.

Use `dbStatement::setHostVar` to populate the parameters.

```
void proc (dbStatement *theStatement,
          const char *procStatement,
          const int numParameters);
```

theStatement	A previously instantiated <code>dbStatement</code> .
procStatement	A null-terminated string containing the stored procedure statement.
numParameters	Number of parameters in the stored procedure statement. If not specified, then the statement is assumed to have no parameters.

execute

Executes a non-SELECT statement.

```
void execute (dbStatement *theStatement);
void execute (dbStatement *theStatement,
             const int ignoreDbError);
```

fetch

Executes the first fetch for a SELECT statement, which fills up the fetch buffer and positions the cursor at the first record. Calls to `dbChannel::nextObject` and `dbChannel::skipObject` automatically fetch more data from the database as needed.

```
void fetch(dbStatement *theStatement);
```

theStatement A previously instantiated `dbStatement`.

close

Closes a statement (cursor). For best performance, the statement should only be “softly” closed.

```
void close(dbStatement *theStatement,
           const int    hardClose);
```

theStatement A previously instantiated `dbStatement`.

hardClose Do an actual database cursor close.

nextObject

Retrieves the next column from a fetch statement.

```
dbObject *nextObject(dbStatement *theStatement);
int       nextObject(dbStatement *theStatement,
                    dbObject     *returnObject,
                    int           maxLength);
```

theStatement A previously instantiated `dbStatement`.

returnObject Where to return the data. If not a CHAR data type, it must match the data type of the object to fetch.

maxLength Maximum length allowed by `returnObject`.

Returns

1. The `dbObject`. This is allocated from the heap and the caller is responsible for freeing it.
2. Status (length):
-1 — NULL data.

- 0 — All is fine
- > 0 — Data truncated to this length.

skipObject

Skips objects from a fetch statement.

```
void skipObject(dbStatement *theStatement,
               int          numColsToSkip,
               const int    stopOnColZero);
```

- | | |
|----------------------|---|
| theStatement | A previously instantiated dbStatement. |
| numColsToSkip | The number of columns to skip. If stopOnColZero is false, then this wraps around. |
| stopOnColZero | Forces a stop at the first column of the next fetch if numColsToSkip results in a wrap. |

getBlob

Gets a BLOB (binary large object) from a fetch statement. You must call nextObject before calling getBlob. The nextObject call will return the total length of the blob or clob column.

```
unsigned int getBlob(dbStatement *theStatement,
                   int          column,
                   char          *buffer,
                   unsigned int  bufferLength);
```

- | | |
|---------------------|--|
| theStatement | A previously instantiated dbStatement. |
| column | The zero-based column index of the column containing the BLOB. |
| buffer | Destination buffer. |
| bufferLength | Destination buffer length. |

Returns

The length of the BLOB portion fetched. A zero indicates that there is no more BLOB data to fetch from this column.

putBlob

Puts (sends) a BLOB to the database. The data is not stored in the database until the dbChannel::execute has been performed.

```
void putBlob(dbStatement *theStatement,
            int          isBinary,
            char          *buffer,
            unsigned int  bufferLength);
```

getRowsAffected

Returns the number of database rows affected by the last INSERT, UPDATE or DELETE operation. For all other operations, this function returns a ZERO.

```
int getRowsAffected() const;
```

getCode/getMessage

Returns the last error code and error message.

```
int getCode() const;
char *getMessage() const;
```

set/getIn/OutDateMask

Sets or retrieves the default DATETIME format masks to use. See Appendix A for details.

```
void setInDateMask(const char *mask);
char *getInDateMask();
void setOutDateMask(const char *mask);
char *getOutDateMask();
```

mask A null-terminated string containing the format mask.

setUserErrorMap

Sets an array of database error codes that when encountered, throw a `dbExceptUser` rather than a `dbExceptDB`.

```
void setUserErrorMap (int            *errors,
                      const int    numErrors);
```

errors An array of integers containing error codes.

numErrors Number of error codes in `errors`.

userError

Returns true if an error code is in the error code map. If not, returns false.

```
int userError(const int errorCode);
```

stmtCacheOn/Off

Turns the dbStatement cache on or off. The default is off. Once turned on, a dbStatement that is destroyed is placed in the internal cache. When instantiating a new dbStatement (with a SQL statement), the cache is searched for a match on the SQL statement. If there is a match, the cache is used. If the dbStatement is instantiated without a SQL statement and is later assigned a SQL statement, the cache is not searched.

When turning off the cache any dbStatements in the cache are destroyed.

```
void stmtCacheOn();  
void stmtCacheOff();
```

throwException

This is the only function that throws any dbExceptions.

It is a virtual function that allows users to supply their own functions if so needed. The user-supplied function must throw its own exception or call throwException to avoid unexpected behavior.

```
virtual void throwException(const int exceptionType,  
                           const char *errorMessage);
```

exceptionType	Type of exception to be thrown.
errorMessage	Error message to set in exception.

connect

Connects to an actual database.

```
void connect (char *connectString);
```

```
connectString VORTEX connect string.
```

release

Releases (disconnects) from a database.

```
void release();
```



Chapter 3

dbChar/dbBinary

class dbChar:public dbObject

Encapsulates character or binary data. This is a sub-class derived from dbObject.

Data Members

None.

Constructors

The first constructor is used to create a NULL object. The last constructor is used to allocate a buffer without initializing it to any specific data (useful for dbChannel::nextObject).

```
dbChar();
dbChar(dbChar *value);
dbChar(char *value);
dbChar(void *value,
        const int length);
dbChar(const int length);

dbBinary();
dbBinary(dbBinary *value);
dbBinary(unsigned char *value,
        const int length);
dbBinary(const int length);
```

Destructor

Frees the internally allocated data buffer.

```
~dbChar();

~dbBinary();
```

Member Functions

See dbObject.



Chapter 4

dbDate

class dbDate : public dbObject

Encapsulates an internal datetime. The internal format is compatible with Oracle's internal datetime. This is a sub-class derived from `dbObject`.

Data Members

None.

Constructors

The first constructor is used to create a NULL datetime.

Destructor

Default.

Exceptions

The following exceptions can be thrown by `dbDate` functions:

`dbException`

`dbDate()`: Invalid `inDateMask`

`dbDate()`: Invalid parameters

Member Functions

See `dbObject` for additional functions.

timestamp/today

Set the value to the current date. Additionally, `timestamp` sets the current time.

toChar

Convert a datetime to a null-terminated string.



Chapter 5

dbDescriptor

class dbDescriptor

A simple class that describes the output column.

Data Members

```
int type;  
int length;  
int nullsAllowed;  
int precision;  
int scale;  
char name[31];
```

type Output column's data type. See `dbDescriptor.hxx` for values.

length Output column length.

nullsAllowed True if nulls are allowed.

precision Precision of a numeric output column.

scale Scale of a numeric output column.

name Name of the output column, null-terminated.

Constructors

Internal.

Destructor

Default.

Member Functions

isLargeObject

Return true of the column is a BLOB.

```
int isLargeObject() const;
```



Chapter 6

dbException

class dbException:public xmsg

The base class for all VORTEX++ exceptions. The derived classes are:

- dbExceptionFatal — Fatal VORTEX++ errors.
- dbExceptionVortex — Fatal VORTEX errors.
- dbExceptionError — Non-fatal errors in the VORTEX++ libraries.
- dbExceptionDB — Database errors.
- dbExceptionNet — Network errors.
- dbExceptionUser — User-defined database errors. See `dbChannel::setUserErrorMap`.

Catching Messages

When you catch a dbException, be sure to get a **reference** to the dbException instead of a **copy** of it. Since xmsg contains a message pointer, each copy of dbException points to the same message which leads to problems when the dbException is destroyed.

Example (note the "&")

```
try {
...
}
catch (dbException& e) {
    cout << "\n***** ERROR *****\n" << e.why() << "\n";
}
```

Data Members

```
int errorType;
```

Each time an exception is thrown, the type of the exception is stored in this information variable. The values are (see `dbChannel.hxx`):

1. typeException
2. typeExceptError
3. typeExceptDB
4. typeExceptUser
5. typeExceptFatal
6. typeExceptVortex

7. typeExceptNet

```
int errorCode;
```

Each time an exception is thrown, the error code of the exception is stored in this information variable.

Constructors

```
dbException(dbChannel *theChannel,  
            const char *errorMessage);
```

theChannel A previously instantiated dbChannel.

errorMessage A null-terminated string containing the error message.

Member Functions

getType/getCode

Return the error type and code associated with the exception.

```
int getType() const;  
int getCode() const;
```

getChannel

Return the dbChannel that threw the exception.

```
dbChannel *getChannel() const;
```




Chapter 7

dbInt

class dbInt:public dbObject

Encapsulates an integer. This is a sub-class derived from dbObject.

Data Members

```
int data;
```

data The actual integer. It is safe to manipulate this directly.

Constructors

The first constructor is used to create a NULL integer.

```
dbInt();  
dbInt(dbInt *value);  
dbInt(const int value);  
dbInt(const int value,  
      const int datatype);  
dbInt(void *value,  
      const int length);
```

value The value.

datatype For setting BLOB integer type.

length For integers of length less than sizeof(int).

Destructor.

Default.

Member Functions

See dbObject.



Chapter 8

dbNumber

class dbNumber:public dbObject

Encapsulates an internal number. The internal format is compatible with Oracle's internal numbers. This is a sub-class derived from dbObject.

Data Members

None.

Constructors

The first constructor is used to create a NULL number.

```
dbNumber();
dbNumber(dbNumber      *value);
dbNumber(const         int value);
dbNumber(const unsigned int value);
dbNumber(const         long long value);
dbNumber(const unsigned long long value);
dbNumber(void         *value,
               const   int length);
```

Destructor.

Default.

Member Functions

See dbObject for additional functions.

toInt

Convert a number to an integer. If no parameter is given and the number is NULL, then a 0 is returned.

```
int toInt();
int toInt(int   ifNull);
int toInt(int   *status);
```

ifNull Value to return if the number is NULL.

status Status of conversion:

- -2 NULL
- -1 truncated or negative
- 0 ok
- 1 overflow

toInt8

Convert a number to a long long integer. If no parameter is given and the number is NULL, then a 0 is returned.

```
long long toInt8();  
long long toInt8(long long ifNull);  
long long toInt8(int *status);
```

ifNull Value to return if the number is NULL.

status Status of conversion:

- -2 NULL
- -1 truncated or negative
- 0 ok
- 1 overflow

toUInt

Convert a number to an unsigned integer. If no parameter is given and the number is NULL, then a 0 is returned.

```
unsigned int toUInt();  
unsigned int toUInt(int ifNull);  
unsigned int toUInt(int *status);
```

ifNull Value to return if the number is NULL.

status Status of conversion:

- -2 NULL
- -1 truncated or negative
- 0 ok
- 1 overflow

toUInt8

Convert a number to an unsigned integer. If no parameter is given and the number is NULL, then a 0 is returned.

```
unsigned long long toUInt8();  
unsigned long long toUInt8(long long ifNull);  
unsigned long long toUInt8(int *status);
```

ifNull Value to return if the number is NULL.

status Status of conversion:

- -2 NULL
- -1 truncated or negative
- 0 ok
- 1 overflow

toFloat/toDouble

Convert a number to a 4- or 8-byte floating point value. If no parameter is given and the number is NULL, then 0.0 is returned.

```
float toFloat();  
float toFloat(float ifNull);
```

```
double toDouble();  
double toDouble(double ifNull);
```

ifNull Value to return if the number is NULL.

sqrt

Return the square root of a number.

```
dbNumber *sqrt();
```



Chapter 9

dbObject

class dbObject

The base class for all VORTEX++ data objects. The derived classes are:

- dbBinary
- dbChar
- dbDate
- dbInt
- dbNumber

Data Members

See sub-classes.

Constructors

See sub-classes.

Destructor.

See sub-classes.

Member Functions

getDatatype/getLength

Return data type and length of the object. See `dbDescriptor.hxx` for datatype values.

```
int getDatatype() const;  
int getLength()   const;
```

getAddr

Return the address of the actual (raw) data.

```
virtual void *getAddr() const;
```

toChar

Convert a data object to a null-terminated string. Individual sub-class may have additional conversion functions.

```
virtual char *toChar(char      *bufp,  
                    const int  maxBufLength,  
                    const int  mustCopy,  
                    char      *ifNull);
```

bufp	Address of target character buffer. If <code>mustCopy</code> is false then this buffer is not used for character or binary objects.
maxBufLength	Allocated size of the target buffer.
mustCopy	Forces the use of the target buffer.
ifNull	A null-terminated string that is returned if the data object is NULL.

copyObject

Make a copy of a data object.

```
virtual DbObject *copyObject();
```



Chapter 10

dbStatement

class dbStatement

Encapsulates a SQL statement.

Data Members

None.

Constructors

The 3 last constructors are convenience constructors that simply add a call to `dbChannel::sql`.

```
dbStatement (dbChannel *theChannel);
dbStatement (dbChannel *theChannel,
             const char *sqlStatement);
dbStatement (dbChannel *theChannel,
             const char *sqlStatement,
             const int numParameters);
dbStatement (dbChannel *theChannel,
             const int isProc,
             const char *sqlStatement,
             const int numParameters);
dbStatement (dbChannel *theChannel,
             const char *sqlStatement,
             const int numDimensions,
             const int numParameters);
```

theChannel	A previously instantiated <code>dbChannel</code> .
sqlStatement	A null-terminated string containing the SQL statement.
isProc	Indicates that the <code>sqlStatement</code> is a procedure.
numParameters	Number of parameters in the SQL statement. If not specified then the statement is assumed to have no paramters.
numDimensions	Number of dimensions of paramters in the SQL statement. If not specified and <code>numParamters</code> is specified, the value defaults to 1.

Destructor

A "hard" close on the underlying database cursor is performed before freeing all working space.

```
~dbStatement ();
```

Exceptions

The following exceptions can be thrown by dbStatement functions.

dbExceptError

Invalid method for setting stored procedure parameters

Row n out of bounds (range: 0 - m)

Column n out of bounds (range: 0 - m)

Cannot mix copyParam() with prior setParam()

Cannot mix setHostVar(), copyParam(), and setParam()

Index n out of bounds (range: 0 - m)

Only valid for setting stored procedure parameters

Member Functions

These functions are listed in `dbStatement.hxx`.

hasBlobs

Indicates that the statement contains BLOBs. When fetching data this ensures that only one row at a time is fetched from the database.

```
void hasBlob();
```

execute

Execute a non-SELECT statement. Convenience functions for `dbChannel::execute`.

```
void execute();
void execute(const int ignoreDbError);
```

fetch

Execute the first fetch for a SELECT statement. Convenience function for `dbChannel::fetch`.

```
void fetch();
```

close

Close a statement (cursor). Convenience function for `dbChannel::close`.

```
void close(const int hardClose);
```

getBlob

Get a BLOB from a fetch statement. Convenience function for `dbChannel::getBlob`.

```
unsigned int getBlob(int column,
```



```
char *buffer,
unsigned int bufferLength);
```

putBlob

Put (send) a BLOB to a database for storage. Convenience function for `dbChannel::putBlob`.

```
void putBlob(int isBinary,
char *buffer,
unsigned int bufferLength);
```

nextObject

Retrieve the next column from a fetch statement. Convenience function for `dbChannel::nextObject`.

```
dbObject *nextObject();
int nextObject(dbObject *returnObject,
int maxLength);
```

skipObject

Skip objects from a fetch statement. Convenience function for `dbChannel::skipObject`.

```
void skipObject(int numColsToSkip,
const int stopOnColZero);
```

copyParam

Copy a parameter to the internal parameter buffer. An actual copy of the parameter is made.

```
void copyParam(int row,
int column,
int value);
void copy Param(int row,
int column,
dbObject *value);
```

row The zero-based destination row index.

column The zero-based destination column index.

value The parameter.

setParam

Make a reference to the parameter in the internal parameter buffer. The caller is responsible for freeing the parameter, if its necessary.

```
void setParam(int row,
```

```
int column,
dbObject *value);
```

row	The zero-based destination row index.
column	The zero-based destination column index.
value	The parameter.

setHostVar

Make a reference to the parameter in the internal parameter buffer. The caller is responsible for freeing the parameter, if it's necessary. The first version is used for setting parameters in regular SQL statements. The second version is only used for stored procedure statements.

```
void setHostVar(int index,
               int datatype,
               int length,
               void *variable,
               short *indicator);
```

```
void setHostVar(int index,
               int datatype,
               int length,
               int items,
               int input,
               int output,
               char *name,
               void *variable,
               short *indicator);
```

index	The zero-based parameter index.
datatype	The variable's data type.
length	The variable's length.
items	1 = scalar > 1 = array
input	Is an input variable.
output	Is an output variable.
name	A null-terminated string containing the variable's name.
variable	Address of the variable.
indicator	Address of the indicator variable. You can specify a NULL.

numOutputCols

Return the number of output columns in the SELECT statement.

```
int numOutputCols();
```

getColDesc

Return a description of an output column. It is the caller's responsibility to free the `dbDescription`.

```
dbDescriptor *getColDesc(int column);
```

column The zero-based column index.

endOfScan

Return true if an "end of scan" has been reached on a SELECT statement.

```
int endOfScan() const;
```

getChannel

Return the `dbChannel` to which the statement belongs.

```
dbChannel *getChannel() const;
```

A

address
 actual data 26
 allocating
 database cursors 8
 see also "setting"
 API
 explained 7
 associating
 stored procedures 12

B

base class
 exceptions 7
 objects 7
 binary
 indicating type of data 15
 binary large object
 see BLOB
 BLOB 19
 putting 14
 retrieving 29
 setting datatype 22
 storing 30
 BLOB (binary large object) 14
 buffer
 getBlob 14
 putBlob 15
 setting internal 8
 bufferLength 14
 putBlob 15

C

caching
 dbStatement 16
 cancelling
 database request 10
 catching
 messages 20
 CHAR data type
 returning 13
 class dbChannel 8
 class dbChar
 public dbObject 17
 class dbDate
 public dbObject 18
 class dbDescriptor 19
 class dbException
 public xmsg 20
 class dbInt
 public dbObject 22
 class dbNumber
 public dbObject 23
 class dbObject
 dbObject 26
 class dbStatement 28
 classes
 dbChannel 7
 dbDescriptor 7
 dbException 7
 dbObject 7
 dbStatement 7

close 29
 soft 13
 close command 13
 closing
 cursors/statements 29
 codes
 database error 15
 returning for errors 15
 column
 describing output
 getBlob 14
 columns
 description of output 32
 processing 8
 retrieving 30
 retrieving 13
 returning number 31
 command
 commit/rollback 11
 executing 10
 command code 10
 command parameters
 listing 10
 commandCode 10
 commandParms 10
 commands
 close 13
 execute 12
 fetch 13
 getBlob 14
 getCode/getMessage 15
 getRowsAffected 15
 nextObject 13
 proc 12
 putBlob 14
 set/getIn/OutDateMask 15
 setUserErrorMap 15
 skipObject 14
 sql 12
 stmtCacheOn/Off 16
 throwException 16
 userError 15
 commit command 11
 connect 16
 connection
 encapsulating 7
 convenience functions
 close 29
 execute 29
 fetch 29
 getBlob 29, 30
 nextObject 30
 skipObject 30
 converting
 data object to string 26
 datetime 18
 number to unsigned int 24
 number to unsigned long long
 24
 copying
 data objects 27
 parameters 30
 copyObject 27

copyParam 30
 current date
 setting 18
 cursor
 logical 7
 cursor command 10
 cursors
 allocating database 8
 close 29
 closing 13
 logical 8

D

data
 address 26
 fetching rows 29
 data object
 converting 26
 data objects
 copying 27
 database
 cancelling request 10
 error codes 15
 releasing 8, 16
 returning number of rows 15
 target for SQL 12
 database connection
 see connection
 database cursors
 allocating 8
 database errors
 ignoring 13
 datatype
 getting 26
 setting for BLOB 22
 date
 setting current 18
 DATETIME 15
 datetime
 converting to string 18
 encapsulating 18
 dbChannel 8, 32
 dbChar 17
 dbDate 18
 dbException 20
 dbNumber 23
 dbObject 22
 dbStatement 28
 caching 16
 describing
 output column 7
 see also encapsulating
 description
 of output column 32
 dropping tables 13

E

encapsulating
 connection 7
 SQL statement 7
 endOfScan 32
 errorMessage 16

- errors
 - database codes 15
 - ignoring database 13
 - returning code/message 15
 - exceptions
 - base class for 7
 - storing type 8
 - throwing 16
 - exceptionType 16
 - execute 29
 - execute command 12
 - executing
 - non-SELECT statement 12
 - executing command 10
- F**
- fetch 29
 - fetch command 13
 - fetch statement
 - skipping objects 14
 - fetchBufferSize 8
 - fetching
 - data 13
 - format mask
 - setting or getting 15
 - freeing
 - parameter 30
 - functions
 - cancel 10
 - close 29
 - command 10
 - connect 16
 - copyObject 27
 - copyParam 30
 - endOfScan 32
 - execute 29
 - fetch 29
 - getAddr 26
 - getBlob 29
 - getChannel 21, 32
 - getColDesc 32
 - getDatatype/getLength 26
 - getType/getCode 21
 - hasBlobs 29
 - isLargeObject 19
 - nextObject 30
 - numOutputCols 31
 - putBlob 30
 - release 16
 - setHostVar 31
 - setParam 30
 - skipObject 30
 - sqrt 25
 - timestamp/today 18
 - toChar 18, 26
 - toFloat/toDouble 25
 - toInt 23
 - toInt8 24
 - toUInt 24
 - toUInt8 24
- G**
- getAddr 26
 - getBlob 29
 - getBlob command 14
 - getChannel 21, 32
 - getCode/getMessage command 15
 - getColDesc 32
 - getDatatype/getLength 26
 - getRowsAffected command 15
 - getting
 - BLOBs 29
 - getType/getCode 21
- H**
- hardClose 13
 - hasBlobs 29
- I**
- ignoreDbError 13
 - instance
 - connection 7
 - integer
 - unsigned 24
 - internal buffer
 - setting 8
 - internal parameter buffer 30
 - interrupting
 - wild query 10
 - isBinary 15
 - isLargeObject 19
 - isProcedure 28
- L**
- length
 - dbDescriptor 19
 - listing
 - command parameters 10
 - logical cursor 7
 - logical cursors
 - maximum 8
 - long long
 - unsigned 24
- M**
- mask
 - set/getIn/OutDateMask 15
 - maxColumns 8
 - maxDbCursors 8
 - maxLength 13
 - maxLogicalCursors 8
 - message
 - returning for errors 15
 - messages
 - catching 20
- N**
- name
 - dbDescriptor 19
 - nextObject 13, 30
 - nextObject command 13
 - non-SELECT
 - executing statement 12
 - non-SELECT statements 29
 - nullsAllowed
 - dbDescriptor 19
 - number
 - converting to int 24
 - converting to long long 24
 - numColsToSkip 14
 - numDimensions 12
 - numOutputCols 31
 - numParameters 12
- O**
- objects
 - base class for 7
 - Oracle format
 - datetime 18
 - statement 12
 - output column
 - describing
- P**
- parameters
 - copying 30
 - setting 31
 - SQL statement 12
 - stored procedure 12
 - precision
 - dbDescriptor 19
 - pre-fetch data
 - buffer 8
 - proc command 12
 - procedures 28
 - processing
 - columns 8
 - procStatement 12
 - putBlob 30
 - putBlob command 14
- Q**
- query
 - interrupting 10
- R**
- read/write transaction 11
 - release function 16
 - releasing
 - database 8
 - request
 - canceling database 10
 - retrieving
 - columns/objects 30
 - retrieving column 13
 - returning
 - number of database rows 15
 - returnObject 13
 - rollback command 11
 - rows
 - returning number of 15
- S**
- scale

- dbDescriptor 19
- SELECT statement 29
- set/getIn/OutDateMask
 - command 15
- setHostVar 31
- setParam 30
- setting
 - logical cursor max 8
 - parameters 31
- setUserErrorMap command 15
- sizeof(int) 22
- skipObject 30
- skipObject command 14
- soft close 13
- sql command 12
- SQL statement
 - associating 12
 - encapsulating 7
 - parameters 12
- sqlStatement 12
- sqrt 25
- startUpdateTrans 11
- statement
 - close 29
- stmtCacheOn/Off commands 16
- stopOnColZero 14
- stored procedures 31
 - associating 12
- storing
 - BLOBs 30
- string
 - conversion to 26

T

- tables
 - dropping 13
- theChannel 28
- theStatement 10, 12, 13
- throwException command 16
- timestamp/today 18
- toChar 18, 26
- toInt 23
- toInt8 24
- toUInt 24
- toUInt8 24
- transaction
 - starting 11
- transactions
 - starting 11
- type
 - dbDescriptor 19
- typeofLastException 8

U

- userError command 15

V

- VORTEX++
 - definition 7

W

- wild query

- interrupting 10
- wrapping fetched data 14

X

- xmsg 20